Executive Summary

The main objective of the ADAPTEAM project is to investigate and develop new techniques for Agent Adaptation in multi-agent systems. Such techniques must deal with the problems of team representation for agile organizational changes, collaboration among teammates for best knowledge sharing and task distribution, monitoring and detecting deficiencies in team performance, and rectifying and improving team knowledge and organizational structures. These techniques are best for agents that must perform complex tasks in environments that are dynamic and uncertain.

To solve the above challenging research problems, the ADAPTEAM project has made substantial progress in the past three years (07/97-03/01), and accomplished the following milestones:

1. Developed a graph representation for agent teams, where an adaptable organization is represented by roles (nodes), role-relationships (edges), and role-assignment (task distribution). With this representation, the process of team adaptation can be formalized as a process of searching for the best graph in a graph space where the evaluation criteria are based on the team performance.
2. Developed a negotiation technique for detecting and resolving conflict and inconsistent beliefs among agents. This investigation has inspired a new research project called DYNAMITE in the Autonomous Negotiating Targets (ANT) program in DARPA/ITO.
3. Developed a distributed technique for monitoring team performance by individual agents. In this technique, the monitoring problem is classified as a problem of Socially Attentive Monitoring, and a distributed algorithm is developed and shown to be sound and complete. This is a property that has not yet been shown for any centralized monitoring algorithms. A PhD dissertation has written by Dr. Kaminka and its title is "Execution monitoring in multi-agent environments", which can be obtained in http://www.cs.cmu.edu/~galk/Publications.
4. Developed a distributed technique for dynamic task re-allocation and experimented this technique in the formalization of distributed constraint satisfaction problems. The developed algorithm can dynamically modify the structure of an organization by changing roles, role-relationship, and role-assignment. Performance of this algorithm has demonstrated that an organization can be dynamically adapted to team performance.
5. Invented a biological inspired technique called "Digital Hormones" for solving the scalable and distributed control problem for self-reconfigurable systems. This technique has a number of unique properties similar to the biological hormone counterparts, and has been successfully applied to metamorphic robotics systems. A US Patent Disclosure (USC File# 3157) has been applied for this technique.

The personnel supported by this project include the faculty members Dr. Wei-Min Shen (PI) and Dr. Milind Tambe (Co-PI), and graduate students: Zhun Qiu (07/97-12/98, completed MS in USC Computer Science), Behnam Salemi (07/97-03/99), Hyuckchul Jung (01/99 to 08/99), and Gal Kaminka (09/99-06/01, complete PhD in USC Computer Science).

The publications stemming from this research include:

- Shen, WM., B. Salemi, P. Will, Hormone-Based Communication and Cooperation in Metamorphic Robots, (submitted) to IEEE Transactions on Robotics and Automation, 2001.
- Salemi, B and WM Shen, Dynamic and Distributed Task Re-Allocation: An initial investigation. (submitted) to the 7th International Conference on Intelligent and Autonomous Systems, 2001.
- Kaminka, G., Pynadath, D., Tambe, M. Monitoring Deployed Agent Teams, International Conference on Autonomous Agents, 2001.
- Salemi, B., WM. Shen and P. Will, Hormone Controlled Metamorphic Robots, in proceedings of International Conference on Robotics and Automation, Korea, 2001.
- Kaminka, G. and Tambe, M. 2000. Robust agent teams via socially attentive monitoringJournal of Artificial Intelligence Research (JAIR) Volume 12:105-147.
- Kaminka, G., Execution Monitoring in Multi-Agent Environments, PhD Dissertation, University of Southern California, May 2000.
- WM. Shen, B. Salemi and P. Will, Hormone for self-reconfigurable robots, in the proceedings of the 6th International Conference on Intelligent Autonomous Systems, IOS Press, pp, 918-925, 2000.
- WM. Shen, Y. Lu and P. Will, Hormone-based control for self-reconfigurable robots, in the proceedings of International Conference on Autonomous Agents, Spain, 2000.
- Castano, A., WM. Shen and P. Will, CONRO: Towards Deployable Robots with Inter-Robot Metamorphic Capabilities, Autonomous Robots Journal, Vol. 8, No. 3, pp. 309-324, Jul 2000.
- Tambe, M. and Jung, H., The benefits of arguing in a team. AI Magazine, 2000.
- Kaminka, G., and Tambe, M., I am OK, You're OK, We're OK: Experiments in distributed and centralized socially attentive monitoring. In Proceedings of the International Conference on Autonomous Agents. May, 1999.
- Qiu, Z., Tambe, M., and Jung, H., Towards flexible negotiation in teamwork. In Proceedings of the Second International Conference on Autonomous Agents. May, 1999. (Poster paper)
- Kaminka, G. and Tambe, M. 1999. I'm OK, You're OK, We're OK: Experiments in Centralized and Distributed Socially Attentive Monitoring. In proceedings of the International conference on Automonomous Agents, Agents'99.
- Kaminka, G. A., 1999. Execution Monitoring and Diagnosis in Multi-Agent Environments. In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL. (Doctoral Consortium).
- Shen, WM., R. Adobbati, J. Modi, and B. Salemi. Purposeful Behavior in Robot Soccer Team Play. The RoboCup99 Workshop in the International Joint Conference of Artificial Intelligence. (A Poster Paper). August, 1999.

- Will, P, A. Castano, W. Shen. Robot modularity for self-reconfiguration, in Sensor Fusion and Decentralized Control in Robotic Systems II, edited by G.T. McKee and P. S. Schenker. Proceedings of SPIE Vol. 3839, 236-246, 1999.

- Kaminka, G. and Tambe, M. 1998 What is wrong with us? Improving robustness through social diagnosis Proceedings of the National conference on Artificial Intelligence (AAAI) .

- Kaminka, G. A.; Tambe, M., and Hopper, C. M. 1998. The Role of Agent-Modeling in Agent Robustness. In AI Meets the Real-World: Lessons Learned (AIMTRW-98), Stamford, CT, September 1998.

- Shen, WM., J. Adibi, R. Adobbati, B. Cho, A. Erdem. H. Moradi, B. Salemi, and S. Tejada. Building Integrated Mobile Robots for Soccer Competition. In Proceedings of International Conference on Robotics and Automation. Leuven, Belgium. May 1998.

- Kaminka, G., and Tambe, M. What is wrong with us? Improving robustness through social diagnosis. In Proceedings of the National Conference on Artificial Intelligence (AAAI). August, 1998.

- Shen, W.M., J. Adibi, R. Adobbati, B. Cho, A. Erdem. H. Moradi, B. Salemi, and S. Tejada. Toward Integrated Soccer Robot Team. AI Magzine. Fall, 1998.

- Qiu, Z. and Tambe, M. Flexible negotiations in teamwork: extended abstract. In M. desJardine (editor), AAAI FALL Symposium on Distributed Continual Planning. October, 1998.

- Qiu, Z. and Tambe, M. Towards Argumentation-based Collaborative Negotiation: A preliminary report. In International workshop on multi-agent systems, Mass. Institute of Technology, October, 1998.

- Wang, X.J and Shen, W.M., Coordination via Negotiation using Pareto Rationality. Poster in the International Joint Conference on Artificial Intelligence, 1997.

Final Performance Report

ADAPTEAM:
Teaming and Information-Sharing among Adaptive Battlefield Agents

USAF, Air Force Office of Scientific Research
Award Number: F49620-97-1-0501
Period of Performance: 07/01/97 – 03/31/01

Wei-Min Shen (PI) and Milind Tambe (Co-PI)
University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695
Phone: 310-822-1511
Fax: 310-822-0751
{shen,tambe}@isi.edu

1 OBJECTIVE

The main objective of the ADAPTEAM project is to investigate and develop new techniques for Agent Adaptation in multi-agent systems. Such techniques must deal with the problems of team representation for agile organizational changes, collaboration among teammates for best knowledge sharing and task distribution, monitoring and detecting deficiencies in team performance, and rectifying and improving team knowledge and organizational structures. These techniques are best for agents that must perform complex tasks in environments that are dynamic and uncertain. In this research project, we have focused our efforts on the following fundamental research issues for agent team adaptation:

1. Compared to individual agent adaptation, what are the special aspects for team adaptation? How to represent the concept of team organization so that it is effective and efficient for adaptation?
2. How do agents share information among themselves so that they can detect, diagnose, and resolve conflicts and inconsistencies among their knowledge bases? Does negotiation play any critical role in this process, if so, how?
3. How do individual agents monitor the team performance so that any deviation from the team plan can be detected timely, accurately, and in a distributed manner?
4. How do teammates dynamically redistribute tasks among themselves according to their capabilities, workloads, and team performance in the process of problem solving?
5. How can team adaptation be accomplished without any fixed leaders so that no individual agent failures can paralyze the entire team, and that reorganization can be scaled up to large systems without requiring agents to have global knowledge about the organization? For this task, what can we learn from biological systems where cells seem to accomplish many complex tasks without global knowledge?

2 STATUS OF EFFORT

To solve the above challenging research problems, the ADAPTEAM project has made substantial progress in the past three years (07/97-03/01), and accomplished the following milestones:

1. Developed a graph representation for agent teams, where an adaptable organization is represented by roles (nodes), role-relationships (edges), and role-assignment (task distribution). With this representation, the process of team adaptation can be formalized as a process of searching for the best graph in a graph space where the evaluation criteria are based on the team performance.
2. Developed a negotiation technique for detecting and resolving conflict and inconsistent beliefs among agents. This investigation has inspired a new research project called DYNAMITE in the Autonomous Negotiating Targets (ANT) program in DARPA/ITO.
3. Developed a distributed technique for monitoring team performance by individual agents. In this technique, the monitoring problem is classified as a problem of Socially Attentive Monitoring, and a distributed algorithm is developed and shown to be sound and complete. This is a property that has not yet been shown for any centralized monitoring algorithms. A

PhD dissertation has been written on the subject, and the thesis can be obtained in http://www.cs.cmu.edu/~galk/Publications.

4. Developed a distributed technique for dynamic task re-allocation and experimented this technique in the formalization of distributed constraint satisfaction problems. The developed algorithm can dynamically modify the structure of an organization by changing roles, role-relationship, and role-assignment. Performance of this algorithm has demonstrated that an organization can be dynamically adapted to team performance.

5. Invented a biological inspired technique called "Digital Hormones" for solving the scalable and distributed control problem self-reconfigurable systems. This technique has a number of unique properties inspired from the biological hormone counterparts, and has been successfully applied to metamorphic robotics systems. A US Patent Disclosure (USC File# 3157) has been applied for this technique.

3 ACCOMPLISHMENTS AND NEW FINDINGS

3.1 Properties and Representations of Team Adaptation (GFY98)

In order to accomplish tasks and missions that require collaborative effort in a real-world situation that is dynamic and uncertain, a group of individual agents must adapt and learn as a team. Although the notion of "team learning" clearly implies more than a collection of individual learning agents, the precise difference between team learning and single-agent learning remains an open problem.

In pursuing our objective to build adaptive agent teams, we have analyzed many learning architectures, and proposed a definition of team learning based on the notion of "roles" and "assignments". A role is defined as responsibilities in service of a team plan, and an assignment assigns members of a team to the roles. For example, in a team plan of leader-follower formation flight of two URAVs (Unmanned Reconnaissance Air Vehicles), the role of the leader may be specified as following a path, while the role of the follower is specified as following the leader. With these two roles, there are two possible assignment for the two URAVs.

Given these concepts, team learning can be performed in at least four aspects to continually improve team performance: (1) to improve an individual's ability to perform a role; (2) to change member/role assignment; (3) to modify roles themselves (such as create, delete, change roles); and (4) to modify the team plan by changing the interactions between roles. In our two URAV example, the first two aspects are obvious. For the third aspect, the team may learn to change the leader role to maintain contact with the follower. For the fourth aspect, the team may learn to change the distance to be maintained between two URAVs. In contrast, non-team learning (or single-agent learning) covers only the first of the four aspects of team learning.

To operationalize this definition, we have developed a new representation called Partial Organizational Structure (POS) based on the concept of Cooperation Structure proposed by d'Inverno, Luck and Wooldridge (IJCAI 1997). A POS is acyclic graph where nodes are roles and edges are relationship between roles. Each node is also labeled with an agent identifier to specify the role assignment. A POS graph is "partial" in the sense that it may only provides a local view of the team activities. In a context of team, each agent produces and executes its own POS. Based on the feedback at the team level, all agents will modify their own POS such that the composition of all POS will converge towards a better performance at the team level. The novel aspect of POS is that it separates roles from agents and allows richer relationships between roles. Furthermore, it uses team performance as a measurement of the structure and permits adaptation in a distributed fashion. To address the problem how a set of POS can be modified to produce a better global measurement, we are also investigating Pareto Rationality as a negotiation strategy.

We are currently implementing this POS framework and testing it on three domains: manufacture scheduling where agents may dynamically change their roles according to the output of an assembling line, marching-band formation where agents have to learn role assignment and role relationship in order to improve the quality of the formation, and helicopter engage attack where agents must synchronize their activities. Limited applications are also considered in the RoboCup domain, where a team of agents must perform in a highly dynamic, uncertain, and adversarial environment.

3.2 Reconcile Conflicts during Collaboration (GFY98)

A major issue in multi-agent teamwork is detecting, diagnosing and resolving conflicts and inconsistencies. We have divided up this problem in two phases: (i) monitoring and diagnosing conflicts; and (ii) collaborative negotiations to resolve conflicts. Guided by our previous experience in multi-agent implementations in complex, real-world domains (e.g., battlefield simulations), we have pursued novel approaches in addressing both these problems. In pursuing these approaches, we have built on our previous work on "model-based teamwork". Such models are essentially common-sense, first-principles rules of teamwork, that provide flexibility in agent behaviors in teamwork.

Conflicts arise in teamwork due to a variety of reasons. In particular, each agent in a multi-agent domain has (i) a local view of the environment, or (ii) a local interpretation based on a local context of the events around it, or (iii) local actions that cannot possibly consider all possible global interactions with other agents. As an example, in battlefield simulations, if a message was lost due to radio-interference, the sender would continue to believe that the message was sent, and the receiver would continue to believe that the message was never sent, leading to a conflict of beliefs.

With respect to monitoring and diagnosis of such conflicts, the novel aspect of our approach is its use of social comparison. This is a complementary approach to standard monitoring techniques where users provide monitoring conditions. In particular, here an agent compares own behavior with those of its teammates, to detect differences. (Here, other agents' behaviors are here often inferred via plan-recognition based on observations of low-level actions.) Behavior differences, while not guarantees of conflicts, triggers a detailed diagnosis that can either explain the difference as legitimate, or else a possible conflict. This novel approach to monitoring has been implemented in the context of battlefield simulations, in a system called SAM (socially-attentive monitoring).

With respect to negotiation, our initial approach was to borrow from game-theory, often used in multi-agent environments, although in the context of self-interested (i.e., self-utility-maximizing) agents. However, this approach failed for a variety of reasons. First, solutions appropriate for self-interested agents are many times inappropriate for a team of agents, where the collective utility is more important than any individual utility. Second, game-theoretic approaches focus on global frameworks or rules of encounter (e.g., vickery auction mechanism) that ensure particular desirable behaviors on part of the agents, such as non-deceptive bids in auctions. However, they do not focus on building agents that can negotiate effectively. More importantly, in teamwork settings, conflicts often center on conflicts of beliefs. Thus, teammates often need to be persuaded to change their beliefs. This issue is not addressed in game-theoretic settings; indeed, due to the possibility of deception, such persuasion is discouraged in self-interested agents.

We have instead followed an alternative approach, that we call negotiation via argumentation. Essentially, agents explain or argue their positions to their teammates. Teammates may accept such explanations or counter-argue, and via this process of argumentation agents may arrive at a resolution of their conflict. Argumentation is a well-studied topic in philosophy, and we are building on Toulmin's argumentation schema (Toulmin, 58). A key problem in such argumentation is devising the rules via which agents may argue with each other. Here, a novel aspect of our work is to rely on our previous research model-based teamwork. In particular, common-sense rules of teamwork from teamwork models are used as a basis for generating arguments. Another novel aspect of our work is that it takes real-time negotiation costs into account. Thus, agents will not engage in protracted negotiations, if the cost of negotiations exceeds its benefits. Based on this approach to negotiation, we have begun to build a system called CONSA, and are currently applying CONSA in battlefield simulations.

3.3 Detecting and Resolving Conflicts during Collaboration (GFY99)

In GFY99, we have built on our research from GFY'98, and continued to focus on issues of detecting and resolving conflicts in multi-agent systems, particularly in agent systems built to operate as teams. Recent advances in teamwork, particularly in the form of recently developed teamwork models, that explicitly outline agents' commitments and responsibilities in teamwork, have significantly improved the flexibility in teamwork coordination and communication. However, this research has so far not fully addressed the challenge of resolving conflicts within a team, because team members are by design, not anticipated to enter into such conflicts.

Yet, as agent applications advance to meet the requirements of scale and autonomy, inter-agent conflicts become increasingly inevitable. For instance, while autonomously reacting to dynamic events, agents may be unable to obtain relevant information from others, thus unintentionally interfering in others' actions or plans. Similarly, agents' faulty sensors may provide them with conflicting information, or their local contexts may lead them to conflicting inferences. While such conflict resolution is a difficult challenge in general, it is even more problematic in teams, precisely because intra-team conflicts are not anticipated.

As mentioned in the last year's report, the challenges here are two-fold: (i) detecting conflicts in teams; (ii) resolving detected conflicts. In detecting conflicts, we have been building on the socially-attentive monitoring approach that we mentioned in last year's report. In this approach, agents use plan recognition to monitor their teammates and detect conflicts by noticing differences with teammates in areas where agreement is assumed mandatory (Plan recognition is used to avoid reliance on communications). Since in our work, socially-attentive monitoring relies on plan-recognition, it must address the problem of uncertainty in plan-recognition. In particular, agents' actions can be ambiguous in that several interpretations are possible, and thus inferences based on them may not be accurate. Here, we have experimented with several different plan-recognition algorithms for socially-attentive monitoring. A major result of our investigation is that a simpler plan-recognition algorithm, that does not explicitly represent the ambiguity in plan-recognition can lead to better socially-attentive monitoring (when employed in a distributed fashion), than a more complex plan-recognition algorithm that does explicitly represent ambiguity but operates in a centralized fashion. We have been able to show this result experimentally, and prove the result theoretically. Thus, the distributed algorithm is shown to be sound and complete, but the centralized one is either incomplete or unsound. We have also been able to identify a key property of agent teams that enable the distributed socially-attentive monitoring technique to provide guarantees of soundness and completeness.

In terms of resolving conflicts in teams, teams, we are developing a system called CONSA (COllaborative Negotiation System based on Argumentation). In recent years, there has been a growing interest in argumentation-based conflict-resolution in multi-agents (although outside of the context of teamwork research). In this approach, agents negotiate by providing arguments (which may be justifications or elaborations) in support of their proposals to one another. CONSA builds on this past work, significantly advancing the state of the art in several ways, as outlined below.

One key insight in CONSA is to fully exploit the benefits of argumentation in a team setting. Several novel sets of ideas stem from this insight. First, CONSA can turn the tables on conflict resolution, by casting it as a team problem. That is, when an agent detects a conflict, it argues for the establishment of a common team goal to resolve this conflict. Subsequently, all relevant team members participate in this conflict resolution via argumentation. Thus, the team members explicitly recognize conflict resolution as a common team goal, accepting compromises in their team's, rather than their own interest. More importantly, all of the recent advances in flexible teamwork --- e.g., based on teamwork models --- are now fully brought to bear to guide agent behavior during conflict resolution. This significantly improves negotiation flexibility. For instance, if a team member privately discovers an event that resolves the current team conflict, it will be automatically committed to informing its team members --- it will not just withdraw privately from negotiations. Additionally, with an explicit common team goal, novel argumentation strategies emerge, e.g., agents will not solely focus on refuting teammates' arguments, but they may even attempt to improve the quality of teammates' arguments.

A second novel idea that exploits the above insight of team argumentation is as follows: Team conflicts are often rooted in past teamwork. To argue effectively about teamwork, agents must be knowledgeable about teamwork. Here, CONSA exploits the general teamwork models mentioned earlier in a novel way, i.e., not as a guide to own behavior during conflict-resolution, but as a source for CONSA's domain-independent argumentation knowledge—so developers need not engineer argumentation knowledge from scratch each time.

CONSA also differs in other important ways from previous work. CONSA is integrated within existing agent teams performing tasks in dynamic, complex environments. Thus, practical issues, such as minimizing the resources consumed in negotiations, meeting real-time deadlines are critical in its implementation. To this end, CONSA performs decision-theoretic cost-benefit analysis of negotiation, so that in some cases, agents may tolerate the conflict to avoid costly negotiations. Argument ordering and pruning is also used for negotiation efficiency.

3.4 Agent Organizational Learning for Better Team Performance (GFY99)

In GFY99, we built on our earlier results and represent Agent Organization (AO) as a set of roles, role relationships, and role assignments, and we had formalized the problem Agent Organizational Learning (AOL) as an adaptive process that changes the three aspects of an organization:

(1) the role structures (such as creation, deletion, and modification of roles);

(2) the relationships between roles;

(3) the assignment of agents to roles.

We have implemented these definitions and experimented the system in several domains. Based on the above definitions, the structure of an organization is not maintained by any single agent but distributed into the set of roles in the organization. Each role uses a Partial Organizational Structure (POS) to represent the part of the organization that it is aware of. A POS is a graph where nodes are roles and edges are role-relationship. A POS graph is "partial" because it only provides some partial information about the organization from a role's local point of view. In addition to POS, a role also contains a Controller for manipulating the POS, and a Communicator to communicate with other roles that it has a relationship with. The assignment between a role and an agent is established by a direct communication link through which the role can issue action commands the assigned agent and receive information from the agent.

Compared to the previous work in Economic Organizational Learning (EOL), the research in Agent Organizational Learning has some unique concerns. In EOL, learning is viewed as a process to select and discriminate a set of "routines" (equivalent to our "roles" at a high level) based on experience (Levitt and March 1988). In AOL, we must adapt not only the roles but also the relationships between roles. Furthermore, in EOL there is no concern about the assignment between routines and actors (for all actors are assumed to be humans), whereas in AOL different role-agent assignment would result in very different organizational performance because agents by design have different capabilities.

We have implemented the above ideas in a JAVA system called ORBLE that can dynamically change an agent organization based on the performance of the organization. The existing ORBEL system can modify role-relationship and role assignment in an organization. (We are actively working on role modification as well). The assumptions we made for the ORBEL system are that (1) the organization has only pair-wise role-relationships, (2) there is no "sabotage" roles in the organization, (3) organizational performance is an additive function of individual performance, and (4) no single agent has the complete knowledge about the organization. The ORBLE system consists of a set of (independent) roles and each role maintains its own POS. In a POS, all role-relationships have a "degree of deviation from expectation" (DDE) which records how well a particular relationship meets its performance criteria. For each role, there is one role-relationship marked as "active", and the objective of ORBLE is to have all roles to converge to a stable set of active relationships that provide the best possible performance at the organizational level. This converge process is guided by the external feedback as follows. Initially, all DDE = 0 for all role-relationships, and each role randomly marks one of its role-relationships as "active." Then, each role will improve its active role-relationship by asking its assigned agent to perform some selected actions. After these actions, the role will re-compute the DDE value for all its role-relationships, and set the highest DDE role-relationship to be active and continue to improve that role-relationship. As this process iterates, the set of active role-relationships will become stable and the performance of the organization will improve.

We have experimented ROBLE in two different domains: a marching band example, and a set of transportation problems in Linear Programming. In the marching band example, a team of agents is to form an organization for marching in a formation. The team performance is measured by the sum of distance discrepancies between adjacent agents with respect to a given constant. (For example, in a given formation, the distance between any two adjacent agents must be maintained at 1 meter.) The agents can move forward and turn left or right, and can measure the distance to any agent in sight. The agents, however, may have different moving speed and may choose different adjacent agents to maintain distances. Since the initial organization may be arbitrary, the initial marching performance could be very poor. Using the ORBLE system, these marching agents can improve their organization (by selecting and maintaining active role-relationships) based on the team performance. In the experiments we have simulated in the JAVE system, the agents always converge to a stable organization for a good marching formation. For example, in one good final organization all agents in a row are aligned to their left agent, and the left-most agent aligns itself to the left-most agent in the front row.

In the transportation domain, we assume that a set of transportation agent must form the most cost-effective organization to transport products from a set of sources to a set of destinations. There is a cost associated with each route between a source and a destination and the performance of this organization is measured by the total cost to ship a set of given products from sources to destinations. We assume that each source is managed by an agent, and so is each destination. Furthermore, each agent only knows its local cost constraints, and each agent performs its actions based on local costs, and communicates with other agents with a set of "wishes". For example, agent X may communicate with Y saying that if "I can ship you N more units of products, then it will save me (thus the entire organization) M units of cost." Starting from a random organization (i.e., a set of randomly selected routes and shipments), the objective of this problem is to find a set of role-relationships (route and shipment) that give the best performance for the entire organization. Notice that these agents are not selfish, and some of them must sacrifice their own benefits in order to gain a better organization performance. Among the experiments we did in this domain, ORBLE is able to find the oprimal organization for some problems (i.e., the result is the same as that computed by a centralized Simplex algorithm when a global view is given). We are currently analyzing these results and investigating if such results can be reached for all problems in this domain.

To change role assignment in an organization, ORBLE uses a simple approach. During the learning process, ORBLE checks if any role's commands have exceeded its assigned agent's capability. For example, a marching agent may be repeatedly asked by its role to increase its speed even though the agent has reached its max speed. Such a case also arises when the agent is damaged and its performance cannot keep up with the commands sent by its role. In these cases, ORBLE will search among all the known agents and seek a more capable agent who is doing a less demanding job, and switch the role assignment between the two.

We are currently extending this organizational learning approach in two major directions. First, we are developing new ideas to allow the learning algorithm to change role structures (such as creating, deleting, composing, and decomposing roles) and deal with more complex learning tasks for role-relationship and role assignment. Second, we are generalizing the algorithm to solve more organizational learning problems in other domains. One very interesting application is self-configurable robots, where a dynamically connected robot modules must work together to change shape and size of the overall robot in order to solve the tasks in hand. Such metamorphic robots are highly desirable in tasks such as fire fighting, search and rescue after an earthquake, and battlefield reconnaissance, where robots must go through unexpected situations and obstacles and perform tasks that are difficult for fixed-shape robots. For example, to maneuver through a difficult terrain, a metamorphic robot may become a snake to pass a narrow passage, grow a few legs to climb over an obstacle, or roll down a slope as a ball. Similarly, to enter a room through a closed door, a self-configurable robot may disassemble itself into a set of smaller units, crawl under the door, and then reassemble itself in the room. To rescue a child trapped deep in rubble in an earthquake, a set of small robots may form a large structure to carry cooperatively an oxygen cylinder that is too heavy for any individual robot. We have recently completed the construction of 20 of these robot modules, and we are designing experiments of organizational learning on these modules.

3.5 Performance Monitoring in Multi-Agent Systems (GFY00)

One of the major successes of the AFOSR grant for FY'00 was the successful doctorol thesis defense of Dr. Gal Kaminka, who was supported by this grant. Dr. Kaminka's PhD thesis was entitled "Execution monitoring in multi-agent environments", which he defened in May 2000. Dr. Kaminka is currently a post-doctoral fellow at the school of computer science at Carnegie Mellon University. A copy of this thesis can be obtained at http://www.cs.cmu.edu/~galk/Publications.

Our research in FY'00, which was part of Dr. Kaminka's thesis, focused on monitoring agent teams. In particular, we focused on the role of plan-recognition in addressing the *monitoring selectivity* problem. While agents in a team must monitor teammates to ensure effective team performance (referred to as *socially attentive monitoring*), monitoring all their teammates all the time can be overwhelming for any individual agent. It is thus critical to develop algorithms that enable agents to selectively monitor teammates, yet provide some guarantees on detecting failures in teamwork. Towards this end, we devised plan-recognition-based distributed monitoring algorithms that provided guarantees of sound and complete failure detection. Interestingly, in these algorithms, agents only monitored some *key teammates*, thus ensuring high monitoring selectivity. We illustrated that such distributed algorithms outperformed much more complex centralized plan-recognition-based monitoring algorithms, which monitored all teammates, and yet were unable to provide similar soundness and completeness guarantees for failure detection. This research was published in the following article: G. Kaminka and M. Tambe, "Robust agent teams via socially attentive monitoring," Journal of Artificial Intelligence Research (JAIR), 12:105-147, 2000.

Further extension of the plan-recognition-based monitoring work focused on monitoring teams based on agents' communications with each other. The key here is to provide information about agent teams' current state based on their on-going communications. This technique is applicable in situations where agents are themselves unable to provide detailed information about their own state to a monitoring agent. For instance, consider deployed agent applications. Here, it is difficult to change existing agent code to cause agents to report their state to a monitoring agent --- indeed legacy code is notoriously difficult to modify. Furthermore, as is well recognized in the literature, agents' continuous communication in all states leads to significant scale-up difficulties (hence, once again, we need to address monitoring selectivity problem). Monitoring based on agents' routine communications avoids such difficulties, and that is the method we adopted. However, routine communications are often sparse—they create significant ambiguities that a monitoring agent must address. We have built a new plan-recognition algorithm to address these difficulties. One key idea to address ambiguities was to exploit the fact that *a team is more than a collection of individuals*. That is, a team of agents will tend to work together, in agreement, on joint goals. The algorithm we developed was implemented and tested on a working application, and it showed on average 84% accuracy in monitoring agents' state. For comparison, a naive algorithm that did not exploit teamwork ideas had a low 10% accuracy in the same application.

3.6 Dynamic and Distributed Task Re-Allocations (GFY00)

Self-organization (also known as adaptive organization, organizational self-design, or organizational learning) is one of the fundamental problems in multi-agent systems. It is about how autonomous agents organize themselves into a collaborative structure and adapt such a structure for better performance. This ability is extremely important for distributed systems that have multiple autonomous agents. One example of such systems would be a group of unmanned air vehicles in a search and rescue mission in an adversarial territory. Any static organization, no matter how cleverly designed beforehand, cannot couple with the dynamics and uncertainties encountered in the real world.

One of the main difficulties in self-organization is that the objective and the feedback are not at the same granularity level. An organization's performance is typically evaluated at the global level, yet its modification is mostly distributed to local and individual agents. Previous research tackles this problem with different assumptions. For top-down approaches such as (So and Durfee 1993), it is common to assume that there is an analytical model for bridging the two levels, and thus the best organization could be found by static analysis of the performance model without execution. This is also true for approaches that are based on agent coalition formation (Shehory and Kraus 1996), where the values of coalitions are assumed to be computable before the execution of the organization. On the other hand, for bottom-up approaches to self-organization such as (Ishida, Gasser, and Yokoo 1992), it is assumed that agents have universal capabilities (i.e., every agent can handle every subtask) and the population of agents can be changed arbitrarily when adapting the organization to the environmental change.

In addition to the granularity discrepancy problem, self-organization is also a very diverse natural phenomenon, and it is difficult to give a coherent and general definition. Many different definitions exist in the literature. For example, (Durfee, Lesser, and Corkill 1989) defines an organization as a set of problem solvers with "information and control relationships." Ishida, Gasser, and Yokoo (1992) describe an organization as a set of production systems with shared variables. (Levitt and March 1988) describe an organization as a set of "routines". (So and Durfee 1993) models an organization as a task dependent structure that includes the task units to be done, the participating (universally capable) agents, an assignment of the tasks to the agents, and a workflow structure dictates the task distribution and result assembly. Most of these definitions, although they provide valuable case studies, are not completely operational for execution.

One way to address the problem of granularity discrepancy and the lack of a general definition is to realize that agents have limited resources and heterogeneous capabilities, and that the feedback from execution must be utilized in searching for a better organization. For this purpose, this paper defines an organization as an assignment of agents to a task-specific *role-graph* and defines self-organization as the optimization of agent assignment in the context of distributed constraint satisfaction problems.

In a role-graph, a *role* node is a set of obligations (or subtasks) for the given task, and a *role-relationship* edge is a commitment between roles to communicate certain types of information (such as subtasks, solutions, actions, or data). This representation separates the role requirement from the agents' capabilities, and makes the assignment of agents to roles an essential task in organization. As we will see later, since agents are allowed to trade their obligations, the assignment task can essentially affect almost all aspects of an organization, including the structure of the role graph and the directions of information flow between agents. With this representation, self-organization is a team-learning process for creating and adapting a role-graph and searching for an optimal assignment of agents to the graph such that a given task can be solved most effectively and efficiently.

3.6.1 Dynamic and Distributed Task Re-Allocation for DCSP

To study self-organization in a domain-independent fashion, it is necessary to ground the research on a rigorous computational foundation that is domain-independent and decomposable among multiple agents. Examples of such foundations include Distributed Constraint Satisfaction Problems (DCSP), Distributed Bayesian Networks, Contract Nets, and Graphical Models (Jordan 1998). In this paper, we shall focus on DCSP to investigate the feasibility of the approach.

A Constraint Satisfaction Problem (CSP) is commonly defined as assigning values to a list of variables V from a respective list of domains D such that a set of constraints C over the variables is satisfied. For example, we can define an example CSP_1 as follows: $V=(x_1,x_2,x_3)$, $D=(\{1,2\}, \{2\}, \{1,2\})$, and $C=\{(x_1 \neq x_3), (x_2 \neq x_3)\}$. Then a solution for CSP_1 is $(x_1,x_2,x_3)=(2, 2, 1)$. A distributed CSP is a CSP in which V, D, and C are distributed among multiple agents. A DCSP is solved if each agent solves its local portion of the CSP and the collection of all local solutions is a solution to the CSP. For instance, we can partition the above example into two parts: $\{V1=(x_1,x_2), D1=(\{1,2\},\{2\}), C1=\{(x_1 \neq x_3)\}\}$ and $\{V2=(x_3), D2=(\{1,2\}), C2=\{(x_2 \neq x_3)\}\}$, and assign them to two agents respectively. Then, a solution to the DCSP is $(x_1,x_2)=(2,2)$, and $(x_3)=(1)$. In the standard DCSP, however, self-organization is not an issue because the assignment between agents and variables are assumed to be given and static.

To generalize DCSP to address the self-organization problem, we introduce that (1) every variable has a set of required capabilities, (2) every constraint has two obligations: the *initiator* and the *accommodator*; and (3) there is a set of heterogeneous agents that collectively possess all the required capabilities. For any given constraint that links two variables, the obligation of the initiator is to select a value for its variable and pass the value to the accommodator. The obligation of the accommodator is to adjust the value of its variable so that it satisfies the constraint with the initiator's value.

Formally, the problem of self-organization for DCSP can be defined as a tuple (V, R, D, C, A), where V is a list of variables, R a list of capabilities required by the variables, D a list of value domains for the variables, C a set constraints, and A a set of agents with heterogeneous capabilities. The goal of this problem is to find an assignment $A \Leftrightarrow (V,C)$ that is both *complete* and *optimal*. An assignment is complete if every variable and every constraint obligation is assigned to a *qualified* agent and no single capability is assigned to more than one variable simultaneously. An agent is qualified for a variable if the agent possesses the necessary capabilities required by the variable. An assignment is optimal if it enables the given DCSP to be solved with the minimal cost among all possible assignments. The cost can be measured in a number of different ways, including the total number of messages sent between agents (or variables), or the sum of computational time consumed by the participating agents.

To illustrate the above definitions, we can extend our CSP_1 example to a self-organization problem $SOCSP_1$ as follows: $V=(x_1, x_2, x_3)$, $R=(\{c_1\}, \{c_2\}, \{c_3\})$, $D=(\{1,2\}, \{2\}, \{1,2\})$, $C=\{(x_1 \neq x_3), (x_2 \neq x_3)\}$, and $A=\{A_1=\{c_1, c_2\}, A_2=\{c_2, c_3\}\})$. Without lost of generality, we assume that each variable requires a unique capability so that capabilities can be represented by variables. With this simplification, R and A in the above example can be represented as $R=(\{x_1\}, \{x_2\}, \{x_3\})$ and $A=\{A_1=\{x_1, x_2\}, A_2=\{x_2, x_3\}\})$.

With the above definition, we can enumerate all possible organizations for a given self-organization problem by matching the qualification of agents with variables and constraints. In our current example, there are eight possible agent assignments (or organizations) and they are listed in Table 1.

Table 1: Possible organizations for *SOCSP₁*

Agent Assignment (Organization)	M_R	M_L
O1: {A1:x1,x2},{A2:x3}, [x1→ x3 → x2]	4.6	1.9
O2: {A1:x1,x2},{A2:x3}, [x1→ x3 ← x2]	4.2	2.0
O3: {A1:x1,x2},{A2:x3}, [x1← x3 → x2]	4.1	2.1
O4: {A1:x1,x2},{A2:x3}, [x1← x3 ← x2]	3.5	1.9
O5: {A1:x1},{A2:x2,x3}, [x1→ x3 → x2]	4.0	2.2
O6: {A1:x1},{A2:x2,x3}, [x1→ x3 ← x2]	3.8	2.0
O7: {A1:x1},{A2:x2,x3}, [x1← x3 → x2]	3.9	2.2
O8: {A1:x1},{A2:x2,x3}, [x1← x3 ← x2]	3.2	1.9

The above organizational notion can be graphically represented in a Distributed Organizational Constraint Network (DOCN). In a DOCN, the nodes are the variables with required capabilities, the edges are the constraints, and the direction of an edge represents the obligation assignments for the constraint pointing from the initiator to the accommodator. A DOCN is complete and optimal if the agent assignment to the elements in the DOCN (nodes and edge directions) is complete and optimal. To illustrate the representation of DOCN, Figure 1(a), 1(b), and 1(c) shows three organizations for *SOCSP₁*: O1, O2, and O8, respectively. Note that O1 and O2 have the same variable assignment, {A1:x1,x2},{A2:x3}, but have different constraint obligation assignment for "x2≠x3". In O1, A1 is the accommodator and A2 the initiator, while in O2, A1 is the initiator and A2 the accommodator. Shown in Figure 1(c), O8 has a totally different variable assignment from O1 and O2; A1 is assigned with x1, and A2 is assigned with x2 and x3.

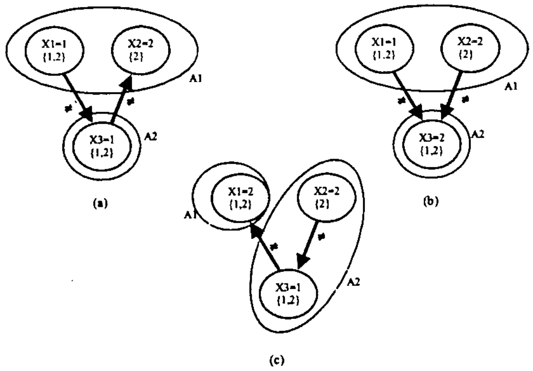


Figure 1: DOCN examples for O1, O2, and O8.

Table 1 also lists the communication cost of each organization for solving the *CSP₁* problem. The second column M_R is the number of remote messages between agents and the third column M_L is the number of local messages within agents. These numbers are obtained by using the Asynchronous Backtracking Algorithm (Yokoo et. al. 1998) to solve *CSP₁*, although other DCSP algorithms could also serve the same purpose. The measurements are the average for 10 trails with the initial value for the variables randomly determined. As we can see, O8 is the best organization for this example. This matches our intuition because x_1 and x_2 are not linked by any constraint so they are assigned to different agents, and x_2 has the most restricted domain so it is assigned as the initiator.

3.6.2 Self-Organization by Heuristic Search

The above definition of self-organization implies that solving the problem requires a search in an enormous space for possible assignments between agents and DOCN elements (nodes and edge directions). This search can be done either exhaustively or heuristically.

The basic idea for exhaustive search is quite simple. One can find the best organization by going through all possible organizations and returning the one that has the best performance. This search is complete because it guarantees to find the optimal organization. But the time complexity of this search makes it impractical to use.

To find a practical solution for self-organization, we may use local search approaches to find an approximate of the best organization by incrementally improving the current organization based on the feedback of solving the given DCSP. To do so, we have to answer two questions: what are the actions for modifying an organization, and when to apply these actions so that they result in improvement. By definition, an organization can be modified by two actions: to trade variables between agents, and to trade constraint obligations between agents. We now discuss them in detail.

Trading Variables between Agents

In an organization for DCSP, an agent assignment actually corresponds to a partition of the given variables. Thus, trading variables between agents means altering the partition of the variables. When a variable x is reassigned from agent A to agent B, the roles and the role-relationships in the organization are also changed. This is because a role is a partition element (a set of variables), and a role-relationship is the set of constraints between two partition elements. By migrating x from A to B, the local links between x and other local variables in A will become remote links between x and A, while the remote links between x and B will become local in B.

When should a variable x be traded from an agent A to an agent B? This depends on the remote communication cost of x with B (i.e., the total number of messages from x to the related variables in B) verses the local communication cost of x in A (i.e., the total number of messages from x to other variables of A). If the former is higher than the later, then it is better to move x from A to B because it will save remote communication cost of the organization. Here, we assume the cost of a remote message between agents is much higher than the cost of a local message within an agent. The above cost information is typically available in the process of solving a DCSP. Each agent will record the number of messages sent and received by its variable, and these numbers are then used to compute the variable's local and remote communication cost.

Thus, to improve an organization based on trading variables between agents, we have the following "variable-trading" heuristic:

- When a variable's remote communication cost is higher than its local communication cost, the variable should be migrated to the remote agent that is qualified for the variable and has the highest remote communication cost with the variable.

Trading Constraint Obligations

Trading constraint obligations between agents or variables is another way to modify an organization. In DOCN, this is equivalent to switching the direction of an edge. In an organization, this action affects the direction of information flow between roles. This action can also affect the performance of an organization because it has been demonstrated in (Armstrong and Durfee 1997) that changes in the priority order among variables can influence the rate of problem solving in DCSP. In this case, switching edge direction is a special case of changing priorities of variables.

To implement this action, however, we have to be careful that no loops of constraint should be formed in the new organization. For this consideration, we assume that every variable is assigned a global priority, and whenever an edge needs to switch direction, the two variables involved in the constraint will exchange their priority values. Notice that although this priority-switching action may cause more changes than switching the direction of a single edge, it does not introduce any new priorities for the variables, thus has less dramatic effects to an organization than the reaction of "nogood" used in (Yokoo, et. al. 1998). This finer modification helps to speed up the rate of finding a solution for DCSP, as we will see in the experimental section. The trigger of this action, however, is the same as the nogood situation in (Yokoo, et. al. 1998). Thus, to improve an organization based on constraint obligation switching, we have the following "priority-switch" heuristic (a relaxed version of constraint-obligation-switching):

- If an agent find a variable x_k that has no consistent value, the agent will switch the priorities between x_k and the inconsistent variable that has the highest priority.

Notice that if both variables are in the same agent, the priority switch is local. Otherwise, it is between agents.

3.6.3 The SOLO Algorithm

In this section, we describe a self-organization algorithm called SOLO for finding the 'best' organization based on the local search heuristics defined above. Similar to many existing DCSP algorithms, the SOLO algorithm allows agents to negotiate by messages to find a solution for a given DCSP. Two types of messages are commonly used. An *ok?* message is sent when an agent is proposing a new value assignment for its local variable. A *nogood* message is sent when an accommodator agent finds it is impossible to adjust its local value assignment to satisfy a constraint. These messages, in the context of self-organization, will be extended to reflect the needs and opportunities for variable trading and priority switching.

SOLO implements the variable-trade heuristic as follows. Whenever an agent *Ai* sends out an *ok?* message for a value update for a local variable x_i, it also indicates its willingness to give away (*giveAway?*) the variable to the receiver. When the *ok?* message is received by an agent *Aj* that has the required capability for x_i, *Aj* will reply to *Ai* with an "*interested*" message. In an ideal situation, among all the "interested" agents, *Ai* should "*release*" x_i to the agent that has the highest remote communication cost with x_i so it can save the most remote communication cost for the organization. In practice, however, x_i is released to the first interested agent because communications are asynchronous and the giver agent cannot wait for all interested parties. Nevertheless, this action tends to gather variables that are heavily depending on each other (i.e., have more traffic) into the same agent, thus reduce the total remote communication in the organization.

The priority-switch heuristic is implemented as follows. Whenever an agent checks its variables and finds a local variable x_k that has no consistent value, it searches for the inconsistent variable x_h that has the highest priority, and then switches the priorities between x_k and x_h. This action is different from Yokoo's heuristic for priority updating (Yokoo, et. al. 1998) because no new priority is created and the resulted change in the organization is much more local.

```
when received (ok?, (Aj, xj, dj, priority, giveAway?)) do
  add (Aj, xj, dj, priority) to agentView;
  if (giveAway? && (hasRequiredCapability(xj)) then
        send(interested, Ai, Aj, xj);
  when agentView and currentAssignments are not consistent do
        checkAgentView;
end do; end do;

when received (nogood ,xj ,nogood) do
  add nogood to the nogoodList
  when (xk, dk, priority) is contained in the nogood
        where xk is not in the neighbors do
        add xk to neighbors,
        add (Ak, xk, dk, prority) to agentView; end do;
  checkAgentView;
end do;

when received (interested, Aj, xi) do
  if xi is in the possessedVariablesList then
        delete xi from possessedVariablesList ;
        send (release, (xi, di, priority)); end if
end do;

when received (release, (xi, di, priority)) do
  add xi to possessedVariablesList;
  annouce the new ownership of xi to the neighbors;
end do;

procedure checkAgentView
  if agentView and currentAssignments are consistent then
        for each xi that has a new value d do
                for each agent Ak that has a constraint with xi do
                        giveAway? = CommCost(xi,Ak)>localCommCost(xi));
        send (ok?, (Ai, xi, d, currentPriority(xi), giveAway?));
  else select $x_k$ from possessedVariablesList, which has the
                highest priority and violating some constraint
                with higher priority variables;
                if no value in $D_k$ is consistent with
                        agentView and currentAssignments then
                        record and communicate a nogood, i.e., the subset
                                of agentView and currentAssignments where
                                $x_k$ has no consistent value;
                        when the obtained  nogood is new do
                                switch the priorities between $x_k$ and the inconsistent
                                variable that has the highest priority in the nogood;
                                $x_k = d$; where $d \in D_k$ and d minimizes the number
                                        of violations with lower priority variables;
                                checkAgentView; end do;
                else $x_k = d$; where $d \in D_k$ and d is consistent with
                        agentView and currentAssignments and minimizes
                        the number of violations with lower priority variables;
                        checkAgentView; end if; end if;
```

Figure 2: The SOLO Algorithm

Figure 2 illustrates SOLO's procedures for receiving messages such as *ok?*, *nogood*, *interested,* and *release,* and for checking local agent views. The lines that are related to organizational modification are underlined for better legibility. We assume the algorithm starts with an initial agent-variable assignment and variable-priority assignment determined randomly. Each agent assigns values to its local variables, and sends *ok?* messages to all related accommodator agents. After that, agents wait and respond to incoming messages. When an *ok?* message about a variable *x* is received, the receiver agent *A* will update its local view and send back an *interested* message if it is capable of and interested in possessing *x*. When an *interested* message for a variable *x* is received, the receiver agent will relinquish the variable (by deleting *x* from its local variable list) and replies with a *release* message. The receiver of the *release* message will add the variable to it local variables list and announce the new ownership by a set of *ok?* messages.

To illustrate the SOLO algorithm in detail, let us consider our *SOCSP$_1$* example again. We assume that the initial organization and values are chosen as in Figure 1(a). Each agent communicates these initial values via *ok?* messages. When A1 sends an *ok?* message to A2 for x1's new value, it also indicates the willingness to give x1 away (because x1's local communication cost in A1 (0) is less than its remote communication cost with A2 (1). When A2 receives this *ok?* message, it updates the *agentView* but is not interested in x1 because it does not have the required capability. After A2 assigns a new value 2 to its local variable x3, it sends an *ok?* message to A1 (for A1 is the accommodator of the constraint x3≠x2). A1 discovers that there is no consistent value for its local variable x2 to satisfy the constraint of x3≠x2, so it performs the following actions. A1 sends a *nogood* message {(x3=2)} to A2, switches the priority value of x2 with x3, selects a new value 2 for x2, sends an *ok?* message to A2 to inform the priority switch and its willingness to give x2 away. At this point, the organization is modified as shown in Figure 1(b). In this new organization, A2 sends out two messages: an *interested* message to A1 for taking x2 (since it has the required capability for x2), and a *nogood* {(x1=1),(x2=2)} message to A1 because it cannot find a consistent value for x3. After these messages, A1 releases x2 to A2 and changes the value of x1 to 2. This is a solution to the given DCSP and the final organization is shown in Figure 1(c).

3.6.4 Experimental Results

This section evaluates the effectiveness of the proposed heuristics and the implemented algorithm. For variable trading, we use a simpler version of SOLO, called SOLO-VT, where only the variable trading heuristic is included. We then compare the performance of SOLO-VT and SOLO-PS with the multi-AWC algorithm (Yokoo and Hirayama 1998) using the distributed 3-color problems.

Given *n* variables, we first generate a random 3-color problem with 2.7*n* links (to ensure the difficulty of the problems). We then generate a set of *m* agents by randomly partitioning the capabilities (variables) into *m* even subsets. If the *n*/*m* is not an integer, then the remaining capabilities are assigned to the last agent. To make sure that agents have overlapping capabilities, we then expend each agent's capabilities by adding extra *p*%, randomly selected different capabilities. Notice that when *p*=0, no agents are able to trade variables, and SOLO-VT is then functionally equivalent to multi-AWC.

Table 2 lists the results of running SOLO-VT with different number of variables (*n*), agents (*m*), and capability overlapping (*p*). Each data point in the table is the average for 50 randomly generated problem instances. The initial values of the variables in these trails are determined randomly. To show the effects of variable trading, we have recorded the number remote and local messages, the cycles needed to solve the problem, and the number of variable trading.

Table 2: The Effects of Variable Trading

n/*m*	MAWC	SOLO-VT		
	p=0	*p*=30	*p*=60	*p*=90
	# *of remote messages*			
10/4	119.0	79.3	68.1	62.0
10/8	265.1	186.7	137.4	200.8
20/8	1004.7	2394.5	1111.7	593.2
20/12	5729.7	3652.9	3038.3	1132.6
30/10	2584.0	2269.4	2490.3	2343.2
30/20	5969.4	7007.3	4236.4	348.9
50/10	2948.9	3152.1	3446.1	3310.4
100/20	6041.7	5920.6	5766.5	5718.1
n/*m*	# *of local messages*			
10/4	29.8	19.1	17.6	21.6
10/8	17.8	25.3	19.6	35.4
20/8	142.2	333.3	189.6	89.6
20/12	412.4	379.8	409.2	143.9
30/10	269.4	239.6	264.7	257.1
30/20	176.7	329.6	293.6	19.4
50/10	324.5	353.0	396.4	395.4
100/20	311.5	329.1	333.3	350.3
n/*m*	# *of cycles for solving DCSP*			
10/4	29.8	19.1	17.6	21.6
10/8	5.8	5.8	4.1	8.2
20/8	20.8	44.6	33.5	16.5
20/12	47.0	47.0	60.2	21.2
30/10	26.7	27.4	31.8	32.1
30/20	21.3	32.4	37.6	3.7
50/10	34.0	39.7	39.8	41.9
100/20	22.8	24.7	26.1	27.0
n/*m*	# *of variable trading*			
10/4	0.0	0.7	1.1	0.5
10/8	0.0	0.0	2.5	2.0
20/8	0.0	0.1	2.8	3.9
20/12	0.0	3.7	5.0	4.9
30/10	0.0	1.3	3.6	5.8
30/20	0.0	7.0	8.0	6.6
50/10	0.0	7.0	8.7	7.4
100/20	0.0	7.2	9.9	10.8

As we can see from the results, as the overlapping capability increases, more variables are traded between agents, less communication is needed between agents, and the rate of converge is faster (less cycles). In general, when agents have choices for what they do, self-organization allows them to solve the problem much more quickly than fixed organization. Interestingly, we notice that the savings in communication does not always go monotonically with the overlapping of capabilities. In the case m/n=20/8, we see an increase of communication at 30% of overlapping, before it goes down again. This may corresponds to some phase transactions in the complexity of self-organization.

Table 3: Comparison of SOLO-PS with multi-AWC

n/m	SOLO-PS	Multi-AWC
20/8	20.9 / 1183.4	21.2 / 1077.4
30/10	25.4 / 2353.3	26.3 / 2548.8
50/10	34.0 / 2408.6	34.2 / 2478.3
100/20	29.8 / 5841.7	30.8 / 6041.7

To evaluate the priority-switch heuristic, we use another version of SOLO, called SOLO-PS, where only the priority-switch heuristic is implemented. Table 3 lists the result of comparing SOLO-PS with multi-AWC for the rate of problem solving. Each data point contains the number cycles/checks to solve the problem. As we can see, SOLO-PS performance is not significantly different from multi-AWC, showing that the heuristic is functionally very close to the priority-updating heuristic used in multi-AWC.

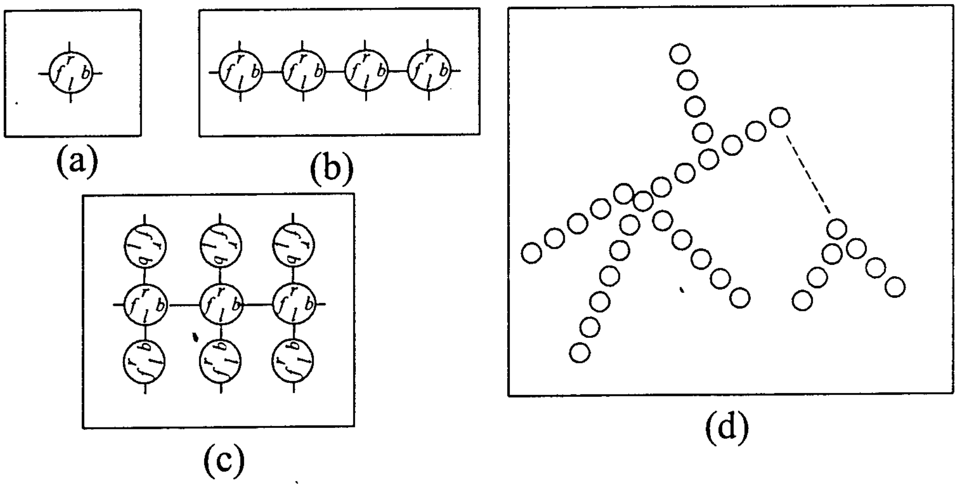
3.7 *Hormone-Based Control for Self-Reconfigurable Systems (GFY01)*

3.7.1 M-Cell Organizations

To apply the concept of hormone to cooperative systems, we envision a set of autonomous agents or robots as an organization of cell-like components. We call these components *m-cells* to emphasize that they are autonomous mechanical/computional systems but they communicate and control each other via hormone-like signals. Abstractly, we will represent any multiple m-cell system as a graph or network of m-cells, with nodes being m-cells, and edges being communication links. Each m-cell has a set of potential communication links that can dynamically connected to other m-cells. We assume that within each m-cell, the communication links can be uniquely identified locally. For example, a CONRO robot module (see http://www.isi.edu/conro) can be represented as an m-cell shown in Figure 3(a) that has four links, named as *f* (front), *l* (left), *r* (right), and *b* (back), respectively.

Thus, a graph for a snake with four self-reconfigurable robot modules can be represented as a graph in Figure 3(b), a 6-legged insect in Figure 3(c), and a system with two separate robots with a remote communication link (dashed line) in Figure 3(d). For a system where all m-cells can broadcast signals to any other m-cells, the network of m-cells will be a completely connected graph. But in general, an organization of m-cells can be an arbitrary graph and m-cells can be different types (i.e., having different number and name of their local communication links).

Given the definition of m-cell organization, "hormones" are defined as signals that are currently active in the system. Computationally speaking, hormones are messages that are "propagating" in the m-cell network. A hormone can trigger different actions at different m-cells because the selection, execution and coordination of actions are determined locally by receiver m-cells. Formally, a hormone is a type of message that has three important properties: (1) it has no destination but propagates in an m-cell organization; (2) it has a lifetime; and (3) it can trigger different actions at different receiving m-cell. The actions caused by a hormone may include modification and relay of other hormones, execution of certain local actions, or just ignoring the received hormone. Different from the conventional message-passing computing paradigms, hormones do not have destination addresses or receiver identifiers, but contain data and action code. When a m-cell receives a hormone message, it will decide its actions completely depending on its local information and knowledge. This total autonomy of m-cells increases the robustness of the organization because even if certain m-cells are damaged, the organization may still function because other m-cells are still in working conditions.


(a)

(b)

(c)

(d)

Figure 3: Examples of M-Cell Organizations

It is worthy to point out that hormones are different from broadcast messages or traditional content-based messages. Hormones are "propagated" signals that may be modified, delayed, or determined along the way of propagating from the source to the rest of the entire global system. In conventional message-passing protocols, messages are sent to certain destinations, and all receivers will obtain the same copy of the same message. Another difference is that when an m-cell receives a hormone from a link (called the *inLink* of that hormone), it must either propagate the hormone to *all* other links (called the *outLinks* of that hormone) or completely ignore it. If an m-cell generates a new hormone, it sends the hormone to all the links it has. In other words, m-cells have no mechanisms to "select" subsets of its local communication links when propagating a hormone. This property is similar to the communication protocols in artificial neural networks, but m-cells do not have fixed I/O links (i.e., every communication link can both receive and propagate hormones), and they have much more local authority than simple artificial neurons.

3.7.2 The Internal Structure of M-Cells

The internal structure of an m-cell system in a hormone-based organization is shown in Figure 4. Each m-cell has a local decision engine that examines the hormone messages received from some of its communication links, and decides if any local actions should be taken on local devices or on these hormones. An m-cell's decisions for actions are totally dependent on the m-cell's local information and knowledge. Since m-cells may have different local state information, the same hormone message may cause different actions at different m-cells. This property of hormone can be exploited to implement a complex action plan as a single hormone. Actions that can be triggered by a hormone may include executing local actions or sensors, modifying local receptors or programs, generating new hormones, or modifying or terminating certain hormones.

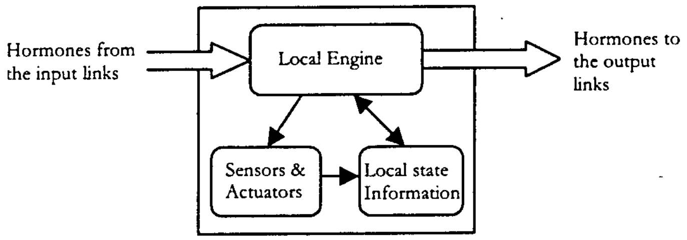


Figure 4: A Hormone-based M-Cell

All m-cells in a hormone-based organization share the same control strategy shown in Figure 4. This uniformity of autonomous cell systems allows adaptations of organization to be totally distributed among the members of the organization. In particular, since any m-cell can communicate with any other m-cell, as long as they have the matching hormones and receptors, new configurations can be negotiated and constructed among m-cells themselves. This property is particularly useful for mobile autonomous systems because they must re-organize themselves whenever the existing configuration becomes ineffective due to the change of the environment or relocations and status changes of some autonomous systems. Furthermore, since programming the local decision engine is a completely local activity and is independent to the number and the state of other cell systems in the organization, this hormone-based control mechanism may be scaled up to very large m-cell organizations.

It is conceivable that hormones may also be used to extend or alter the functions of an m-cell's local engine so that the responses to new and old hormones can be dynamically programmed and altered. Such ability will allow dynamic changes in the functions of m-cells, in a way that is similar to the biological hormones that can enter the cell nucleus.

3.7.3 Types of Hormones

Hormones can be classified and typed in terms of their functions. Each hormone is a list of data fields, and the interpretation of these fields depends on the type of the hormone. Among many types of hormones, two particular types are essentially important for our distributed robot control. They are Action-Specification Hormones (AH) and Synchronization Hormones (SH).

The format of an Action-Specification Hormone is as follows:

(AH, Task, Action, Value, TimeToLive)

The "Task" field specifies the desired task or goal to be accomplished. The "Action" field describes the selected action for the task. The selected action can be an executable local action or a new task to support the given task. The "Value" field provides a place for m-cells to pass local state information and knowledge to other m-cells. The "TimeToLive" records the remaining time before the hormone should be determined.

The purpose of AH is for m-cells to select and propagate actions for tasks. For example, suppose an AH that contains a "move" task is currently active in the system, then m-cells that receive this AH will select an appropriate "gait" for the move task. In this case, a gait is a compound action to support the move task.

After a gait is selected, a new AH can be generated to trigger m-cells to perform appropriate motor positions for the gait. In the new hormone, the Task field contains the selected gait, while the Action field will be filled with motor positions to support the gait task. With this usage, we can see that the structure of Tasks and Actions in hormones are recursive, i.e., an existing action in a hormone can become a Task in another hormone to trigger more actions.

Another important type of hormone for our purpose is the Synchronization Hormone, whose format is as follows:

(SH, Action, AHSet, Value, TimeToLive)

The "Action" field specifies the action to be synchronized among m-cells. The field "AHSet" is a set of AHs that have been received from a particular communication link. (The usage of this field will be explained later.) The Value slot may contain local information and knowledge to be propagated. The TimeToLive field is the same as that of AH. The purpose of Synchronization Hormones is to synchronize actions among m-cells.

Synchronization can have many forms. According to [13] there are thirteen types of temporal relationships among two actions. Two of the types are 'Meets' and 'Starts', which in this proposal are called 'Serial' and 'Parallel' relationships respectively. Serial actions are those that one starts after the other ends and parallel actions are those that start at the same time. These two cases are of our interest since they accomplish almost all synchronization requirements of an m-cell organization. In a later section, we will describe in detail how SHs are used to synchronize actions.

3.7.4 Hormone Propagation

Hormone propagation is a mechanism by which an m-cell communicates with others. In two situations, an m-cell *propagates* a hormone. The first case is when the m-cell generates a new hormone. In this case, the generator m-cell will send (local broadcast) the hormone to *all* its links that are *active*, i.e., connected to a neighbor m-cell. The second case is when an m-cell receives a hormone at a particular communication link and decides to relay it to all other neighbors. In this case, the m-cell marks the receiving link as the *inLink* and will send the hormone to the rest of its active links, called *OutLinks*. Hormone propagation is the only way that an m-cell can communicate with other m-cells. Different from regular message-passing protocols, it is not possible for a module to send a hormone to "some" of its active links. In other words, an m-cell must either propagate a hormone or completely ignore it.

3.7.5 Types of Metamorphic Robot Modules

Metamorphic robots are among the best platform to demonstrate the advantages of hormone-based control methods because no static assumptions can be made for such robots' body shape and size. This section describes how we apply the hormone-based control method to a metamorphic robot. The generalization of these results to cooperative systems in general are made whenever necessary and appropriate.

CONRO modules can be connected together by their docking connectors, called *links*, located at either end of each module. At one end, called *back* (*b* for short), there is a female connector, consisting of two holes for accepting another module's docking pins. At the other end, three male connectors are located on three sides of the module, called *left* (*l*), *right* (*r*) and *front* (*f*). An *active link* is a connector that is connected to another module and the connected module is called a *neighbor*. Each module has a *type*. The module type is determined based on how active links are connected to the neighboring modules. For example, if a module's back is connected to the left of another module and there is no other active links, then it will be of type T5. Each module can determine its type locally by checking to which links of the neighboring modules it is connected. This information is communicated among neighbors. Table 4 lists the 32 distinct types for a CONRO module.

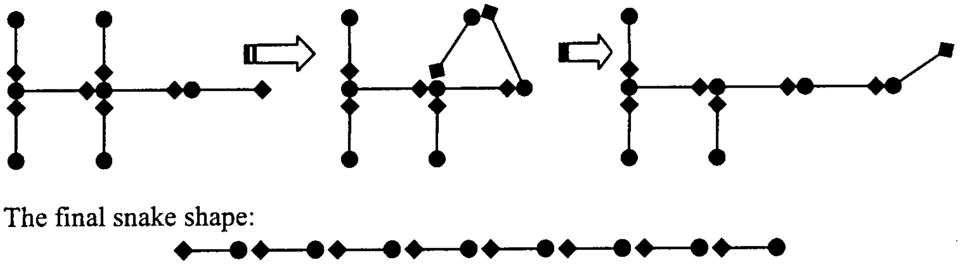
Table 4: The Types of CONRO Modules

	This Module					This Module				
	b	*f*	*r*	*l*	Type	*b*	*f*	*r*	*l*	Type
Connected to other modules					T0	*f*	*b*			T16
	f				T1	*f*		*b*		T17
		b			T2	*f*			*b*	T18
			b		T3		*b*	*b*	*b*	T19
				b	T4	*f*	*b*	*b*		T20
	l				T5	*f*		*b*	*b*	T21
	r				T6	*f*	*b*		*b*	T22
		b	*b*		T7	*l*	*b*	*b*		T23
			b	*b*	T8	*l*		*b*	*b*	T24
		b		*b*	T9	*l*	*b*		*b*	T25
	l	*b*			T10	*r*	*b*	*b*		T26
	l		*b*		T11	*r*		*b*	*b*	T27
	l			*b*	T12	*r*	*b*		*b*	T28
	r	*b*			T13	*f*	*b*	*b*	*b*	T29
	r		*b*		T14	*l*	*b*	*b*	*b*	T30
	r			*b*	T15	*r*	*b*	*b*	*b*	T31

In the later sections, we assume each active link has a buffer for keeping the received hormones. The name of the buffer is a pair of link names, starting with the module's link name. For example, the name of a module's left link buffer connected to the back link of its neighbor will be *lb*. This name will be used to label the received hormones from that link.

3.7.6 A Reconfiguration Example

Before we discuss the general issues of hormone-based control for metamorphic robots, let us consider an example how hormones can be used to guide reconfigurations.



The final snake shape:

Figure 5: Reconfiguration from Quadruped to Snake

Figure 5 illustrates a situation where a CONRO metamorphic robot with seven modules changes from a quadruped (a four legged structure) to a snake. In this figure, a CONRO module is represented as a line segment with two ends: a diamond-shaped end (the back link) and a circle-shaped end (this end has three possible links: the front, left and right). The robot must change from a legged configuration (at the top-left of the figure) into a snake (at the bottom of the figure). To do so, this robot must perform the "leg-tail assimilating" action four times. To assimilate a leg into the tail, the robot first connects its tail to the foot of a leg and then disconnects the leg from the body (shown at the upper part of the figure). Just as in any m-cell organization, each module in the robot determines its role based on its local state information such as its own module type. For example, in this robot, a module is the tail if its type is T2. Similarly, a module knows it is a foot if its type is T5 or T6 and its immediate neighbor is of type T21 or T29.

Using hormones, the entire reconfiguration procedure starts when one (and any one) of the modules generates a reconfiguration hormone LTS (Legs To Snake). This LTS hormone is floating to all modules, but each module's reaction to this LTS hormone will be different and that depends on the receiver's role in the current configuration. For this particular hormone, no module will react except the foot modules, which will be triggered to generate a new hormone RCT (Requesting to Connect to Tail). Since there are four legs at this point, four RCT hormones will be floating in the system. Each RCT carries a unique signature for its sender. No module will react to a RCT hormone except the tail module. Seeing a RCT hormone, the tail model will do two things: acknowledge the RCT by sending out a new TAR (Tail Accept Request) hormone with the signature received in the RCT, and inhibit its receptor for accepting any other RCTs. The new TAR hormone will reach all modules, but only the leg module that initiated the acknowledged RCT will react. It first terminates its generation of RCT, and then generates a new hormone ALT (Assimilating a Leg to the Tail) and starts the required reconfiguration action (see [6] for the details of this compound action). When seeing an ALT hormone, the tail module will terminate the TAR hormone and starts actions to assimilate the leg. After the action is done, the tail module will reactivate its receptor for RCT hormones, and another leg assimilation will be performed. This procedure will be repeated until all legs are assimilated. In Table 5, we list all the hormones involved in the entire reconfiguration procedure.

Table 5: Hormone activities during a reconfiguration

Active hormones	Actions
LTS	Start the reconfiguration
RCT_1, RCT_2, RCT_3, RCT_4	Legs are activated
TAR, RCT_2, RCT_3, RCT_4	The tail inhabits RCT, and leg1 determines RCT_1
ALT, RCT_2, RCT_3, RCT_4	The tail assimilates leg1 and then accepts new RCT
TAR, RCT_2, RCT_4	The tail inhabits RCT, and leg3 determines RCT_3
ALT, RCT_2, RCT_4	The tail assimilates leg3 and then accepts new RCT
TAR, RCT_2	The tail inhabits RCT, and leg4 determines RCT_4
ALT, RCT_2	The tail assimilates leg4 and then accepts new RCT
TAR	The tail inhabits RCT, and leg2 determines RCT_2
ALT	The tail assimilates leg2 and then accepts new RCT
∅	End the reconfiguration

As we can see from this example, hormone-based control has a number of unique features. First, this control procedure can work in many different configurations. In our current example, the procedure will work independent of the number of legs in the system and how long the tail is. Second, the hormones are naturally organized in hierarchical structures. For example, a single LTS can trigger a level of activity managed by the hormones RCT, TAR, and ALT. One ALT will trigger another level of activity for assimilating a leg using another set of hormones (we did not show the details of this level in this example). Third, hormones allow global actions to be totally distributed to individual members. Modules do not have addresses or identifiers, and they have total autonomy in deciding their local actions, generating new hormones, or terminating existing hormones.

3.7.7 A Locomotion Example

Each CONRO module has two degrees of freedom: DOF1 for pitch (up and down) and DOF2 for yaw (left and right). Each DOF has a home position (when the joint is straight), and has two joint limits (when the joint reaches the maximal or the minimal angle). With these two DOFs, a single module can wiggle its body but cannot move. However, when two or more modules connect to form a structure, they can accomplish many different types of locomotion. For example, Figure 6 illustrates a 6-module caterpillar gait. To move forward, each module's DOF1 goes through a series of positions and the synchronized global effect of these local motions is a forward movement of the whole caterpillar (indicated by the arrow).

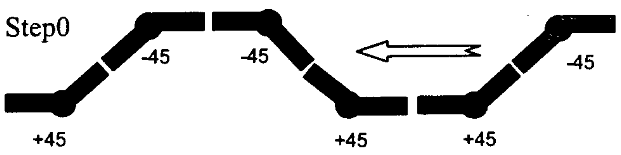


Figure 6: A caterpillar (or nessie) movement

To completely specify this gait, a conventional method is to use a "gait control table" as shown in Table 6, suggested in [14, 15], where each row in the table corresponds to the target position for all DOFs in the configuration during a step. Each column corresponds to the sequence of desired positions for one DOF. The control starts out at the first step in the table, and then switches to the next step when all DOFs have reached their target position in the current step. When the last step in the table is done, the control starts over again at step 0. Table 6 lists the control for the caterpillar movement in Figure 6. As we can see in this table, the six columns correspond to the six module's DOF1 (the leftmost is M1, and the rightmost is M6). The first row in this table corresponds to Step 0 in Figure 6.

Table 6: The gait control table for the caterpillar movement

Step	Module ID for DOF1					
	M1	M2	M3	M4	M5	M6
0	+45°	-45°	-45°	+45°	+45°	-45°
1	-45°	-45°	+45°	+45°	-45°	-45°
2	-45°	+45°	+45°	-45°	-45°	+45°
3	+45°	+45°	-45°	-45°	+45°	+45°

The problem of this conventional gait table method is that it is not designed to deal with the dynamic nature of robot configuration. Every time the configuration is changed, no matter how slight the modification is, the control table must be rewritten. For example, if two snakes join together to become one, a new control table must be designed from scratch. A simple concatenation of the existing tables may not be appropriate because their steps may mismatch. Furthermore, when robots are moving on rough grounds, actions on each DOF cannot be determined at the outset.

To represent a locomotion gait using the hormone idea, we notice that Table 6 has a "shifting" pattern among the actions performed by the modules. The action performed by a module *m* at step *t* is the action to be performed by the module (*m*-1) at step (*t*+1). Thus, instead of maintaining the entire control table, this gait is represented and distributed at each module as a sequence of motor actions (+45°, -45°, -45°, +45°). If a module is performing this caterpillar gait, it must select and execute one of these actions in a way that is synchronized and consistent with its neighbor module.

To ensure the ordered action, each module can use hormones to inform its neighbors what action it has selected, and the neighbors will select their actions according to the desired order. The advantage of this approach over the gait control table is that it will function regardless of how many modules are in the current snake configuration.

3.7.8 Task Specifications

At the present, tasks for a metamorphic robot are either for reconfiguration or for locomotion. They are both specified using the following format:

- A task name;
- An ordered list of actions;
- Synchronization pattern among actions performed by different m-cells;
- A set of constraints on actions selected among neighbor m-cells;

Since actions are ordered in a task, it is possible to define functions like *next()*, or *previous()* on actions, which returns the next or previous action according to the order specified in the task. A constraint has the following form: (<action> <relation> <link-pairs sequence> <action>). The role of constraints is to create patterns of actions among modules by eliminating unsuitable action candidates. There are two symbols that can be used instead of a constraint. The first one is the '+' symbol which means one or more link-pairs and the second is '*' symbol which means zero or more link-pairs.

As an example task specification, a caterpillar gait is defined as:

- Task: Caterpillar-Move
- Actions: An ordered circular list of actions (a_0-a_3), which sets the DOF1 to the specified value: ($a_0 \rightarrow +45°$, $a_1 \rightarrow -45°$, $a_2 \rightarrow -45°$, $a_3 \rightarrow +45°$)
- Synchronization: start at the same time (parallel);
- Constraint: (a_x) *=fb next(a_x)*. This constraint means that if the *f* link of a module *i* is connected to the *b* link of another module *j*, the action of a module *j* should be the next action of module *i*.

Notice that the synchronization between actions performed by different modules is specified as "parallel", which means all modules must start executing their action at the same time. Other types of synchronization include "start after finish (serial)", or "end at the same time". For example, in Table 5, the ALT actions triggered by a LTS must be executed in serial. While the supporting actions for a single ALT must be executed in parallel because all modules involved in a ALT must be active to make a successful docking.

3.7.9 Synchronization Among M-Cells

Synchronization is a general problem for distributed systems. In a master control system [14], synchronization is an operation with a high cost of communication. In a masterless control system [14], it demands an unrealistic assumption that all modules' internal clocks are synchronized.

In a hormone-based control system, solutions to the synchronization problem are naturally suggested by the flexible interpretations of hormones. Since hormones can "wait" at a site for the occurrence of certain events before traveling further, they can be used as tokens for synchronizing events between modules. For example, to synchronize steps in a caterpillar move, a synchronization hormone can be designed to ensure that all modules start moving in a single wave step.

Among many types of synchronization, the *serial synchronization* can be easily realized by hormones because of the way hormones propagate among m-cells. When a serial synchronization among the actions of a task is required, a module will first execute and *complete* its local action before propagating the action hormone to its output links.

The *parallel synchronization*, on the other hand, requires more computational resources. An m-cell cannot execute a local action immediately after the action is selected. It must negotiate with other m-cells to ensure that all actions are started at the same time. For example a spider robot should move its legs simultaneously to perform a successful 'walk' gait. Therefore a mechanism is required to signal modules *when* they should start.

For this purpose, we propose a hormone based synchronization algorithm, which runs on each module in parallel and guarantees the same starting time for all synchronized actions. Generally speaking, for a given task, a module can infer that *all* other modules have selected and are ready to start their actions when it receives the "expected number of hormones" (include both AHs and SHs) from all of its neighbors. The expected number of hormones (ENP) for a neighbor is the number of active links of that neighbor.

In performing a task, when a module selects an action, it generates and propagates an action-specifying hormone, e.g. (**AH**, move, a_1, x, y), to all its neighbors. It then counts the number of received hormones for each link. When a link has received the expected number of hormones, the module sends a synchronization hormone (SH) to the rest of its neighbors. The new hormone contains the received hormones labeled with the link name. For example, if the above action-specifying hormone is received from the link *lb* with ENP=1, then a new synchronization hormone (**SH**, move, *lb*{(**AH**, move, a_1, x, y)}, w, z) will be generated and sent to all other neighbors. The receivers of this synchronization hormone will count it as one received hormone for the appropriate link. Although the content of the AH and SH are not required for the synchronization purpose, they will be kept by modules and will be used in conflict resolution, which will be described in the next section.

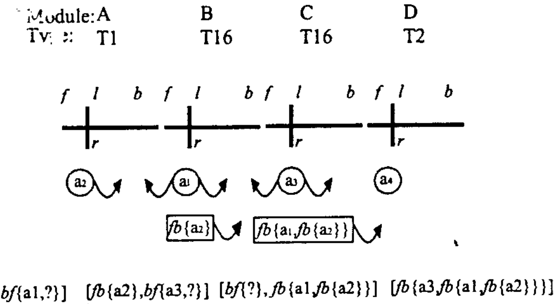


Figure 7: A Snapshot of the Parallel Synchronization

To illustrate this synchronization idea, consider a detailed example of parallel action synchronization of a four-module caterpillar robot shown in Figure 7. For simplicity, it is assumed that the robot is performing only one task, and only actions are represented in the internal representation of received hormones. The type of each module is shown according to Table 4. The number of active links of modules A to D is 1-2-2-1, respectively, which also defines the ENP for each link. For example, the ENP for module A's *bf* link is 2 (for the neighbor B on that link has 2 active links). Figure 7 also shows the hormones that have been propagated. **AHs** are shown in circles and **SHs** are shown in rectangles. Notice that module 'B' ('C') has received the expected number of hormones from 'A' ('B') and has propagated a **SH** to 'C' ('D'). The lower part of the figure shows the contents of the received hormones. The '?' sign represents hormones that are expected. For example, B has two active link buffers: *fb* expected and received 1 hormone a_2, and *bf* expected 2 but received 1 hormone a_3 (so it has a "?"). This synchronization process is complete when 'D' propagates its action a_4. It causes modules 'C', 'B', and 'A' to receive all of the expected hormones and, together with module 'D', start execution of their actions at the "same" time. Here we assume that the time between D sending out a_4 and the completion of propagation of this hormone to all other modules is negligible compared to the execution time of actions. When the synchronization is completed, the contents of received hormones for each module are shown in

Table 7.

Table 7: The contents of received hormones during a parallel synchronization

Module	Internal representation
A	[a_2, [***bf***{a_1, ***bf***{a_3, ***bf***{a_4}}}]]
B	[a_1, [***fb***{a_2}, ***bf***{a_3, ***bf***{a_4}}]]
C	[a_3, [***bf***{a_4}, ***fb***{a_1, ***fb***{a_2}}]]
D	[a_4, [***fb***{a_3, ***fb***{a_1, ***fb***{a_2}}}]]

The above parallel synchronization method works for any configuration of the robot. Compared to the centralized control system with a standard message passing protocol, which requires $O(n^2)$ message hops for each synchronized action (because n messages must be sent to n modules), hormone-based mechanism requires only $O(t^2n)$ hops (Because each module generates t messages and relays $t(t-1)$ messages), where n is the number of modules in the configuration and t is the number of its active links ($1 \le t \le 4$ for CONRO modules).

3.7.4 Conflict Resolution

In the above algorithm it was assumed that when modules select and synchronize their actions, there is no conflict among the selected actions. In reality, since actions are selected independently between modules, it is a possible that actions of two modules violate some constraints in the gait definition. Therefore a conflict resolution phase is required.

The first step in conflict resolution is the *constraint checking*. It examines the selected actions in the internal representation gathered during the action selection phase, such as the ones shown in Table 7. A constraint matches the internal representation of a module if there is an exact match between the link-pairs of a constraint and a sequence of labels connecting two actions in the internal representation. For example, the constraint given in the caterpillar gait, i.e. (a_x) = *fb* *next*(a_x), matches "a_1 *fb* a_2" because $a_x = a_1$ and *next*(a_x)= a_2 according to the action order specified in the gait. However, "a_3 *fb* a_1" does not match the constraint, and one of the actions must be changed.

The decision about which action needs to be changed is based on the TimeToLive (TTL) of the action-specifying hormone. Among the two actions, the "younger" one (i.e., with greater TTL value) will be selected. If conflict is detected before propagation, the selected action will be changed and a consistent action will be propagated. However, in situations that action is already propagated, a conflict-resolution hormone (**CRH**) will be propagated. The format of **CRH** is shown as follows:

CRH(task, **ASH**, constraint , TTL).

A **CRH** contains the conflicting hormone and the violated constraint. In the above example, assuming that the hormone containing a3 has a greater TTL, the generated **CRH** is: **CRH**(move, **ASH**(move, a_3), (a_3 *fb* a_1), *). If a receiving module has the conflicting **ASH** in its internal representation, it will delete that hormone, update the number of received hormones in the receiver buffer, and propagate the **CRH**. Otherwise, the receiving module will ignore the received **CRH**.

When the module whose action is the source of conflict receives the **CRH**, it will select an action that satisfies the constraint included in the **CRH** and generate a new **ASH** containing the new action and propagate the hormone. In the above example, module 'C', which selected a_3, will re-select action a_0 and generate a new **ASH**. If module 'D' receives the new **ASH** before propagating its selected action, it will select a consistent action, which in this case is a_4.

3.7.11 The Basic CELL Algorithm

All activities discussed above are parts of a distributed algorithm called CELL. This algorithm runs locally and autonomously at each m-cell, waiting to receive hormones and perform action selection, synchronization, and execution of local actions. Figure 8 gives the CELL algorithm in pseudo-code.

```
When a hormone (AH, task, action) is received do
{
    selectedAction ← selectNextAction(task, action);
    If (the synchronization of task is "Parallel") {
       propagate(AH, task, selectedAction);
       parallelSynchronization&ConflictResolution(selectedAction);
       execute(selectedAction);
       }
   If (the synchronization of task is "Serial") {
       execute(selectedAction);
       propagate(AH, task, selectedAction);
       }
}
```

Figure 8: The Basic Steps of the CELL Algorithm

3.7.12 Hormone Management

So far we have described hormone-based algorithms for working with single active hormones. To support multiple hormone generation and management, we illustrate briefly in this section how CELL algorithm uses two synchronized hormones to perform both moving and turning in a caterpillar configuration. The "caterpillar-move" gait described above will move the robot along its body and the "caterpillar-turn" gait, as described below, will bend the body to the side. The synchronized combination of these two gaits will generate a circular trajectory. The "caterpillar-turn" gait is defined as:

- Task: Caterpillar-Turn;
- Actions: ($a_5 \rightarrow 0°$, $a_6 \rightarrow 10°$) to set DOF2 to the specified angle;
- Synchronization: Parallel;
- Constraint: (a_x) != $+a_6$, which means there can not be two modules in the robot that perform the a_6 action at the same time. Synchronization between the two gaits (caterpillar-move and caterpillar-turn) are ensured by the following local constraints:
  if (a_x=a_4 & $a_y = a_6$) then $next(a_y) = bf(a_y)$,
  if (a_x=a_3) then $next(a_y) = bf(a_y)$

The last two constraints are used to identify the next action of a module. They ensure that the next action of a module is the action performed by its back neighbor. Along with the constraint that restricts the bending to one and only one module, this gait will shift the bent-action between a unique pair of modules who are performing the caterpillar move $a_4 a_3$.

To effectively use hormones in an adaptive organization, hormones must be properly generated, managed, and terminated. We view hormone generation as another local action of an agent, which can be triggered by an incoming hormone or by some external sensor stimuli. In a decentralized organization, a hormone generator for a particular global task is the temporal leader and coordinator of that task. For example, the tail module is the leader for the task specified in a ALT hormone (see Table 5). Some actions are single events, and others may require a sequence of hormones to be sent out.

For an agent to generate a sequence of hormones, we assume that the agent has the local knowledge about the hormone sequence. One possible representation of such knowledge is a vector **H** of a particular hormone H, along with an index variable H_i. To generate the next hormone in the sequence, the agent simply increments $H_i = mod(H_i+1, |\mathbf{H}|)$, where $|\mathbf{H}|$ is the length of **H**, and then release $\mathbf{H}[H_i]$ into the organization. To ensure that any agent can become the generator for any hormone sequence, we assume that every agent contains all sequences of hormones that have been defined. This can be accomplished dynamically by broadcasting whenever a new sequence of hormone is defined. An agent can become the generator of a hormone sequence in two ways. In the self-promoted case, an agent generates hormones because a local sensor is triggered by some external stimulus. In the instructed case, an agent generates hormones because it received a special hormone trigger message.

Hormones are terminated in a way similar to the way they are generated. That is, an agent can stop producing hormones either by self-promotion (because of an external stimulus) or by a special hormone-stopping message. For example, a hormone may be terminated if certain values are read from a local sensor, or if it receives a special hormone. For example, as shown in Table 5, an m-cell will terminate its RCT when it receives a TAR.

3.7.13 Initial Results of Hormone-Controlled Metamorphic Robots

The concepts and algorithms described above have been implemented and tested in two sets of experiments. The first set of experiments is to apply the algorithm to controlling the real hardware CONRO modules to perform caterpillar move. This implementation has enabled a 4-module CONRO snake to perform autonomous motions with the CELL algorithm running on each module.

In parallel with the hardware implementation of the CONRO robot, we have also used a Newtonian mechanics based simulator, Working Model 3D, to develop the hormone-based control theory, with the objective that the theory and its related algorithms will eventually be migrated to the real robots. Working Model 3D is a three-dimensional dynamics simulation program. Using it, a designer can define objects with complex physical properties, including mass, coefficient of friction, moments of inertia, and velocities. Constraints among objects include rigid joints, revolute joints, and linear constraints, including rods, springs, and dampers. User-defined forces, torques, actuators and motors are also available. In this simulation environment, the CELL algorithm is implemented in Java and running on each simulated module. We have experimented and demonstrated successful locomotion in various configurations, including snakes with different length and insects with different numbers of legs. We are in the process of applying the CELL algorithm to self-reconfiguration actions such as shape changing from a legged configuration to a snake, and vice verse.

4 PERSONNEL SUPPORTED

Faculty:

Wei-Min Shen (~25%)

Milind Tambe (~25%)

Graduate Students:

Zhun Qiu (07/97-12/98, completed MS in USC Computer Science)

Behnam Salemi (07/97-03/99)

Hyuckchul Jung (01/99 to 08/99)

Gal Kaminka (09/99-06/01, complete PhD in USC Computer Science)

5 PUBLICATIONS

- Shen, WM., B. Salemi, P. Will, Hormone-Based Communication and Cooperation in Metamorphic Robots, (submitted) to IEEE Transactions on Robotics and Automation, 2001.
- Salemi, B and WM Shen, Dynamic and Distributed Task Re-Allocation: An initial investigation. (submitted) to the 7th International Conference on Intelligent and Autonomous Systems, 2001.
- Kaminka, G., Pynadath, D., Tambe, M. Monitoring Deployed Agent Teams, International Conference on Autonomous Agents, 2001.
- Salemi, B., WM. Shen and P. Will, Hormone Controlled Metamorphic Robots, in proceedings of International Conference on Robotics and Automation, Korea, 2001.
- Kaminka, G. and Tambe, M. 2000. Robust agent teams via socially attentive monitoringJournal of Artificial Intelligence Research (JAIR) Volume 12:105-147.
- Kaminka, G., Execution Monitoring in Multi-Agent Environments, PhD Dissertation, University of Southern California, May 2000.
- WM. Shen, B. Salemi and P. Will, Hormone for self-reconfigurable robots, in the proceedings of the 6th International Conference on Intelligent Autonomous Systems, IOS Press, pp, 918-925, 2000.
- WM. Shen, Y. Lu and P. Will, Hormone-based control for self-reconfigurable robots, in the proceedings of International Conference on Autonomous Agents, Spain, 2000.
- Castano, A., WM. Shen and P. Will, CONRO: Towards Deployable Robots with Inter-Robot Metamorphic Capabilities, Autonomous Robots Journal, Vol. 8, No. 3, pp. 309-324, Jul 2000.
- Tambe, M. and Jung, H., The benefits of arguing in a team. AI Magazine, 2000.

- Kaminka, G., and Tambe, M., I am OK, You're OK, We're OK: Experiments in distributed and centralized socially attentive monitoring. In Proceedings of the International Conference on Autonomous Agents. May, 1999.
- Qiu, Z., Tambe, M., and Jung, H., Towards flexible negotiation in teamwork. In Proceedings of the Second International Conference on Autonomous Agents. May, 1999. (Poster paper)
- Kaminka, G. and Tambe, M. 1999. I'm OK, You're OK, We're OK: Experiments in Centralized and Distributed Socially Attentive Monitoring. In proceedings of the International conference on Automonomous Agents, Agents'99.
- Kaminka, G. A., 1999. Execution Monitoring and Diagnosis in Multi-Agent Environments. In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), Orlando, FL. (Doctoral Consortium).
- Shen, WM., R. Adobbati, J. Modi, and B. Salemi. Purposeful Behavior in Robot Soccer Team Play. The RoboCup99 Workshop in the International Joint Conference of Artificial Intelligence. (A Poster Paper). August, 1999.
- Will, P, A. Castano, W. Shen. Robot modularity for self-reconfiguration, in Sensor Fusion and Decentralized Control in Robotic Systems II, edited by G.T. McKee and P. S. Schenker. Proceedings of SPIE Vol. 3839, 236-246, 1999.
- Kaminka, G. and Tambe, M. 1998 What is wrong with us? Improving robustness through social diagnosis Proceedings of the National conference on Artificial Intelligence (AAAI) .
- Kaminka, G. A.; Tambe, M., and Hopper, C. M. 1998. The Role of Agent-Modeling in Agent Robustness. In AI Meets the Real-World: Lessons Learned (AIMTRW-98), Stamford, CT, September 1998.
- Shen, WM., J. Adibi, R. Adobbati, B. Cho, A. Erdem. H. Moradi, B. Salemi, and S. Tejada. Building Integrated Mobile Robots for Soccer Competition. In Proceedings of International Conference on Robotics and Automation. Leuven, Belgium. May 1998.
- Kaminka, G., and Tambe, M. What is wrong with us? Improving robustness through social diagnosis. In Proceedings of the National Conference on Artificial Intelligence (AAAI). August, 1998.
- Shen, W.M., J. Adibi, R. Adobbati, B. Cho, A. Erdem. H. Moradi, B. Salemi, and S. Tejada. Toward Integrated Soccer Robot Team. AI Magzine. Fall, 1998.
- Qiu, Z. and Tambe, M. Flexible negotiations in teamwork: extended abstract. In M. desJardine (editor), AAAI FALL Symposium on Distributed Continual Planning. October, 1998.
- Qiu, Z. and Tambe, M. Towards Argumentation-based Collaborative Negotiation: A preliminary report. In International workshop on multi-agent systems, Mass. Institute of Technology, October, 1998.
- Wang, X.J and Shen, W.M., Coordination via Negotiation using Pareto Rationality. Poster in the International Joint Conference on Artificial Intelligence, 1997.

6 INTERACTIONS AND TRASITIONS

6.1 Participation and Presentation at Conferences

Shen, WM., Digital Hormones and Self-Reconfigurable Systems, Invited Talk, ITT/Vanguard Conference on the Future of Software, Los Angeles, 2001. http://www.ttivanguard.com/

Tambe, M. The benefits of arguing in a team, Invited talk, AAAI workshop on "Conflicts in Agents", National conference on Artificial Intelligence, AAAI, 1999.

Tambe, M. Towards flexible teamwork. Invited Seminar, SRI International. 1998.

Tambe, M. Towards flexible teamwork. Invited Seminar, University of Maryland Computer Science Department. 1998.

Shen, WM., Toward Autonomous Agent Team for Real-World Environment. Invited Talk for AI Seminar. Carnegie Mellon University, Computer Science Department, December, 1997.

Tambe, M. Towards flexible teamwork. Invited Seminar, Stanford University Computer Science Department. 1997.

In July 1999, the California Science Center has invited Dr. Wei-Min Shen and USC Soccer-Robot Team (Dreamteam) to give a series demonstrations to the public for 4 weekends. About 500 people have seen the demo, and these events are described by the following two newspaper and magazine articles:

[1] Robot soccer pits machines against machines, *The Christian Science Monitor*, July 29, 1999.

[2] RoboCup: Yet another challenge for robots. *Bridge USA*,' Japanese Info-Tainment Magazine, No. 246, pages 53-56, August, 15, 1999.

6.2 Consultative and advisory functions to other laboratories and agencies

Since GFY 99, we are closely collaborating with Dan Daskiewich in the AFRL Information Directorate on the DYNAMITE project for Autonomous Negotiating Targets (ANT) program supported by DARPA/ITO.

6.3 Transitions

Many important research results and insights initiated in this project, such as multi-agent teamwork, negotiation-based collaboration, and digital hormones, have contributed to the formation of new research projects funded by DARPA:

CONRO: Self-Reconfigurable Robots, 1998-2002, DARPA/MTO Program Manager Dr. Elana Ethridge.

DYNAMITE: Dynamic Negotiating Adaptive Multi-Agent Systems, 1999-2002, DARPA/ITO Program Manager, Dr. Vijay Raghavan.

TEAMCORE: Rapidly Extending and Building Agents to Form Robust, Adaptive Teams, 1999-2001, DARPA/ITO.

7 NEW DISCOVERIES, INVENTIONS, OR PATENT DISCLOSURE

US Patent Disclosure, USC File#3157, Digital Hormones and Self-Reconfigurable Systems, 2001.

8 HONORS AND AWARDS

Shen, WM. et. al., World-Champion Award in 1997 RoboCup, the International Soccer Robot Competition, Nagoya, Japan. August, 1997.

Tambe M. and et. al., Third-Place Award in the First World Cup Simulated Robot Soccer Competition. Nagoya, Japan. August, 1997.

REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-
0492

1. REPORT DATE (DD-MM-YYYY) September 13, 2001	**2. REPORT TYPE** Final Performance Report	**3. DATES COVERED** (From - To) 07/01/1997 – 03/31/2001

4. TITLE AND SUBTITLE

ADAPTEAM: Teaming and Information Sharing among Adaptive Battlefield Agents

5a. CONTRACT NUMBERS

5b. GRANT NUMBER
F49620-97-1-0501

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dr. Wei-Min Shen (PI) and Dr. Milind Tambe (Co-PI)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATIC'N NAME(S) AND ADDRESS(ES)

USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Office of Scientific Research (AFOSR)/NM
801 N. Randolph Street ROOM 732
Arlington, VA 22203-1977

10. SPONSORING/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT

UNCLASSIFIED/UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The ADAPTEAM project is to research and develop new techniques for Agent Adaptation in multi-agent systems. Such techniques must deal with the problems of team representation for agile organizational changes, collaboration among teammates for best knowledge sharing and task distribution, monitoring and detecting deficiencies in team performance, and rectifying and improving team knowledge and organizational structures. During the period of 07/97-03/01, the project has developed a graph representation for agent teams, where an adaptable organization is represented by roles (nodes), role-relationships (edges), and role-assignment (task distribution). A negotiation technique is developed for detecting and resolving conflict and inconsistent beliefs among agents and inspired a new research project for the Autonomous Negotiating Targets (ANT) program in DARPA/ITO. A distributed algorithm for monitoring team performance is developed and shown to be sound and complete (see the PhD dissertation at http://www.cs.cmu.edu/~galk/Publications). Furthermore, a distributed technique for dynamic task re-allocation is developed and experimented in distributed constraint satisfaction problems. Finally, the PI invented a new technique called "Digital Hormones" for distributed control in self-reconfigurable systems. A US Patent Disclosure (USC File# 3157) has been applied for this technique.

15. SUBJECT TERMS

Agent Adaptation, Multi-Agent Collaboration and Negotiation, Self-organization, Digital Hormones.

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Wei-Min Shen
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	34	**19b. TELEPHONE NUMBER (Include area code)** (310) 448-8710